Section 17 Listserver Function

The *listserver* consists of a single process.

> **NOTE:** The *listserver* should not be confused with the *SDB list server*. Also, the *listserver* is sometimes called the *request server*.

The *listserver* process (see Figure 17-1) provides the user with *flight counts* and *flight list* reports on flight information maintained by the ETMS. The report request is made through the *Aircraft Situation Display (ASD),* and the report can be viewed on the screen or obtained in printed form.

The user may obtain other specialized reports: Sector area report (AREA), arrival delay prediction report (ARRD), arrival fix loading report (FIXL), list flight plan report (LIFP), Official Airline Guide (OAG), or schedule database (SDB) report, and time verification report (VT). Flight lists, flight counts, AREA, ARRD, OAG, and VT reports may be obtained for any airport or combination of airports and for sectors, fixes, and Air Route Traffic Control Centers (ARTCCs) as well. FIXL and LIFP reports can be obtained for any one fix or airport. The *listserver* obtains the information for the reports from various ETMS databases. Figure 17-1 shows how the *listserver* communicates with the other ETMS processes.

Design Issue: the Command Line Interpreter

As mentioned above, the *listserver* provides the user with flight information available throughout the ETMS system. A command line interpreter was developed as the user interface to carry out this function. The command line interpreter consists of a *keyword* driven language and a set of syntactical rules on how to use the language. This section of the document describes the main design issues regarding the command line interpreter. One of the issues is understanding the request entered on the command line, and another is how to use the data encoded in the request to obtain the requested information from the ETMS databases. The following discussion describes the steps involved in understanding the request and in filtering the data to produce the desired results. Figure 17-2 summarizes the steps of the process.

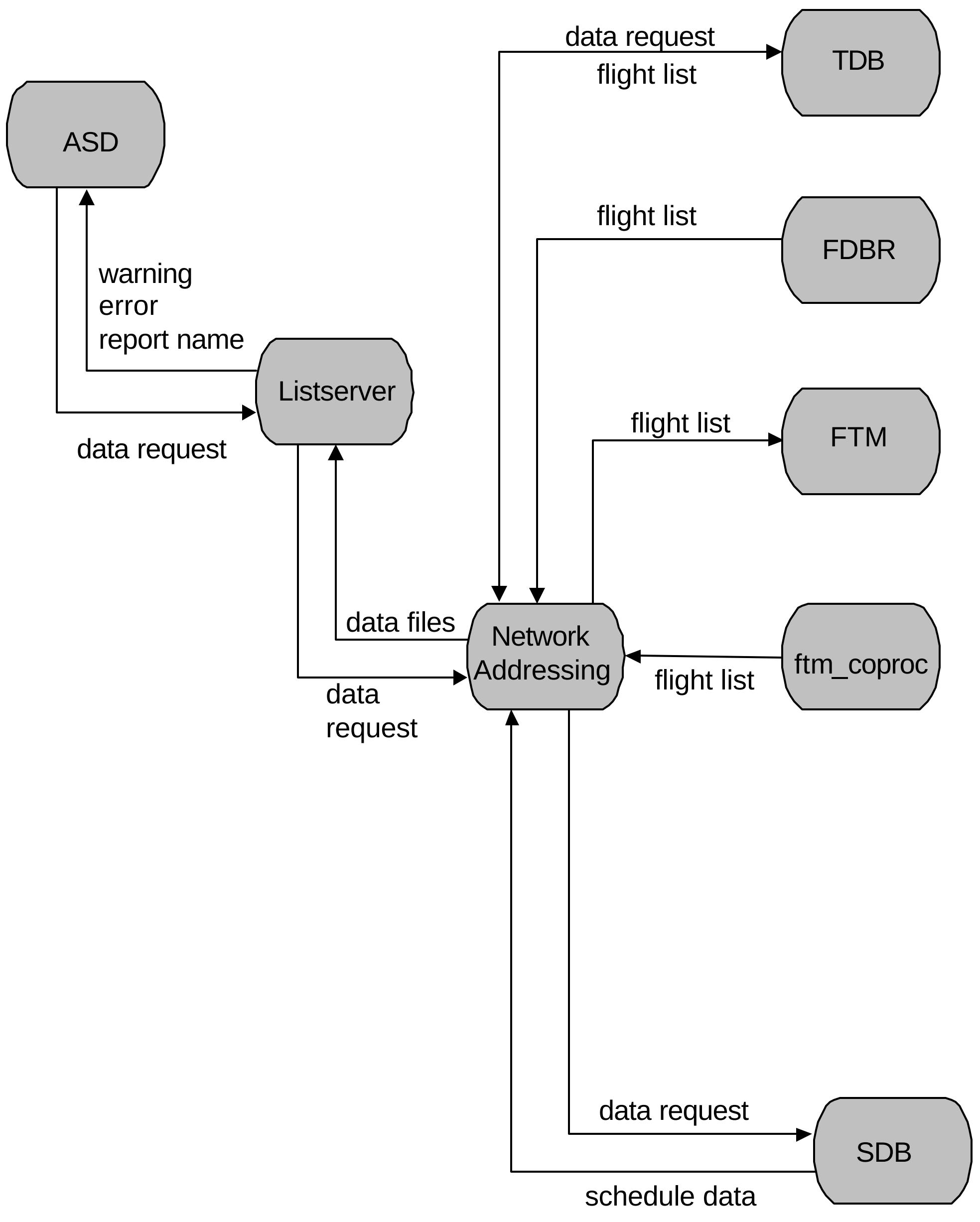

Figure 17-1. Overview of the Listserver's Role in the ETMS

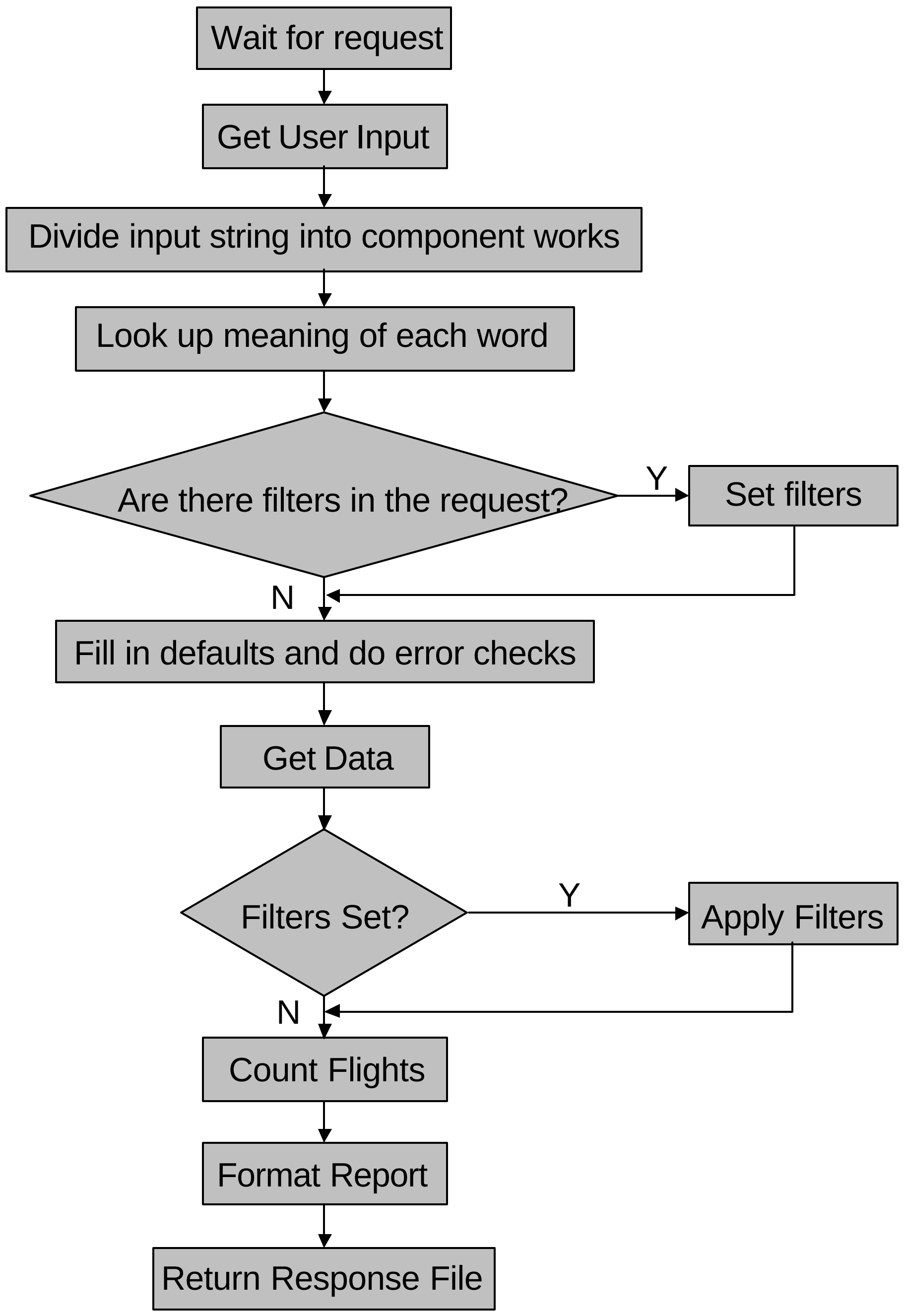
Wait for request
Get User Input
Divide input string into component works
Look up meaning of each word
Are there filters in the request?
Y
Set filters
N
Fill in defaults and do error checks
Get Data
Filters Set?
Y
Apply Filters
N
Count Flights
Format Report
Return Response File

Figure 17-2. Flow Chart of the parsing Algorithm

The following is a typical request which asks for counts for arrivals at **bos** and **jfk** airports on **twa** airline, excluding b727 aircraft, for the time span between 1200 and 1359, inclusively:

```
req a [bos, jfk] airl twa type -b727 1200 to 1400
```

When the input string is received by the *listserver,* it is divided into its component words. Each word is looked up in a *dictionary;* this step associates a *definition* and *value* with each word. The *definition* is used by the parsing process to determine what kind of word it is, and the *value* is used to determine the type of item at hand. The dictionary consists of a hash table created out of a file that contains all the valid words that can appear in a request. These words represent valid FAA designators for airports, Navigational Aids (NAVAIDs), sectors, aircraft types, ARTCCs, and airlines plus all valid keywords accepted by the command line language.

When all the words that appear in a request are successfully identified, they are regrouped into *phrases*. In the context of the *listserver* a *phrase* is a group of words that defines a particular characteristic of the flight information that is being requested. Flight information can be characterized by:

- Demand type - arrivals and/or departures
- Primary location(s) or place(s) of interest
- Secondary location(s) or place(s) of interest
- Time
- Date
- Airlines
- Equipment type, i.e., TYPE B747
- Equipment type prefix and suffix only, i.e., EQUIP R/_/H
- Extended equipment type, i.e., E_TYPE 4R/B747/H
- Aircraft class
- Aircraft category
- User category
- Aircraft remark flag
- Unknown origination airport
- Unknown destination airport

Distinctions can be made between characteristics to be *included* and those to be *excluded*. Besides these basic flight information qualifiers, there are other phrases that characterize the report formatting. Phrases that qualify flight information are called *filters*.

Assuming the demand type is for both arrivals and departures, the primary and secondary places of interest can be defined as follows. The user desires any flight coming from the primary place(s) of interest and going to the secondary place(s) of interest, and any flight going to the primary place(s) of interest and coming from the secondary place(s) of interest. Limiting this search by direction can be done by changing the demand type to A for arrivals or D for departures. D, for departures only, signifies the user wants flights departing the primary place(s) of interest and arriving at the secondary place(s) or everywhere if no secondary place of interest is given. A, for arrivals only, signifies the user wants flights arriving at the primary place(s) of interest that have departed from the secondary place(s) of interest or from everywhere if no secondary place of interest is given. The primary place of interest can have locations to exclude as well. For example, a request of the form REQ A [ZNY -JFK -LGA] [ZBW -BOS] LIST would mean: return to the user a list report with all flights arriving in the N.Y. Center (excluding those arriving at John F. Kennedy or at La Guardia airport) and departing from the Boston Center but not departing from Logan Airport.

The process of grouping the words into phrases is accomplished by applying the various syntactical rules and by the fact that the beginning of a phrase is usually identified by the presence of a keyword. When a keyword is encountered, words following it are grouped, based on their relationship to the keyword as represented by the definitions associated with each word. Let us review the previous request example:

req a [bos, jfk] airl twa type -b727 1200 to 1400

In this case **airl twa** is an airline phrase. This means that airlines are to be specified; **airl** is the keyword, and **twa** is the name of the actual airline. **twa** is the only airline of interest. Similarly, **type -b727** is an equipment type phrase. In this case the user wants to exclude all **b727** type aircraft; note that exclusions are signaled by a minus sign (**-**).

Special components that deserve attention are the primary and secondary places of interest. A request having only a primary place of interest and not limited to either departures or arrivals would mean include anything going to or coming from the specified primary place of interest. If one had a primary place of interest and a secondary place of interest, the meaning takes on the form of anything coming from the primary place going to the secondary place or visa versa. To eliminate the visa versa logic and to specify direction, prefix the two places with an A (for arrivals) or a D (for departures).

When the request is parsed into phrases, a determination is made whether or not there is filtering involved in the request. The *listserver* does not maintain the databases, it is just another user (see Figure 17-1). The *listserver* needs to get data from a database in order to respond to a request. Data is retrieved from a database by giving the database interface module the name(s) of the location(s) for which information is desired, as well as the time range of interest. Therefore, when the *listserver* has received the data it needs to service the request, that data has already been filtered for location and time. In the example above, there are only two filters required: one is for airlines and the other for aircraft type.

Phrases in a sentence represent data characteristics that the user either wants to count or does not want to count. Thus, each phrase can be represented as a Boolean expression. The representation of the request by a Boolean expression provides not only a way to understand what the request means, but a way to use the information encoded in the request to retrieve just the desired information from the ETMS databases.

At this point, it is easier to follow the parsing algorithm by applying it to the example given above. In brief, assume that the request is broken up into its component words, and these have been successfully looked up in the dictionary. The remainder of the parsing is accomplished through the following steps:

- Break down the request into component phrases:
  - o demand type **- a** (arrivals)
  - o Primary location(s) of interest **- bos** and **jfk**
  - o airline(s) to include (**airl**) **- twa**
  - o aircraft type to exclude (**type**) **- b747**
  - o time span **- 1200 to 1400**
- Translate each phrase into a boolean expression:
  - o combine the first two phrases **-** [(**a** = **bos**) or (**a** = **jfk**)]
  - o airline(s) to include **-** (**airl** = **twa**)
  - o aircraft(s) to exclude **-** (**type** <> **b747**)
  - o time span **-** (arrival time > **1200**) and (arrival time < **1400**)
- Connect all these expressions into one single boolean expression which represents the meaning of the request:

  If [(**a** = **bos)** or (**a** = **jfk**)] and (**airl** = **twa**) and (**type** <> **b747**) and (arr time >**1200**) and (arr time < **1400**), then accept flight

  Else reject flight.

The time span phrase is inclusive if the end time minute specified does not end on a 00, 15, 30 or 45 minute. If the time span ends on a 00, 15, 30 or 45 minute then that end time minute will not be included in the report. This was changed to reflect what air traffic controllers expect to get back in a report, based upon the times specified, especially in the case of requesting data that in turn will be fed to *SCDT* programs to determine which flights should be controlled by *EDCT.*

Now that there is a representation of the meaning of the request, the next issue that has to be addressed is how to use the representation to get the data needed and to extract just the information wanted. At this point, the *listserver* associates a filtering routine with each boolean subexpression. That is, there exist declared functions that evaluate just one boolean subexpression. For example, the airline filtering function follows the logic: IF (**airl** = **airline**) then accept flight, ELSE reject flight, where **airl** represents the airline name entered in the request and **airline** would be the appropriate field of the flight record being inspected. Exceptions to this filtering are filtering on EQUIP types and filtering on E_TYPES. Filtering on EQUIP requires checking if the EQUIP type has either the filtered aircraft equipment prefix or the aircraft equipment suffix specified in the flight. Filtering on E_TYPES allows for filtering on multiple fields AND-ed together. For example, filtering on E_TYPE B/L101 would signify one wants to match on all flights that have an aircraft equipment prefix of B/ that are ALSO L-10-11's. Before the *listserver* sends out for data, it sets up the filtering scheme that it will follow once the data is retrieved. The filtering scheme consists of queuing pointers to all the filtering functions that the data must be subjected to. This is illustrated by the filter queue represented in Figure 17-3.

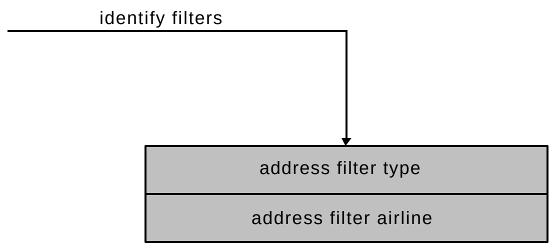

Figure 17-3. Queue of Pointers to Filter Functions

Once the filtering queue is initialized, the *listserver* sends out for the appropriate data. Unless the request is in the form: REQ OAG…, first it will send to the *Traffic Demands Database* (*TDB*) a request for data. *TDB* then returns a file of data, unless the request was a Center list request, in which case *TDB* passes the data file to the *FDBR* process to assist in filtering out restrictions. In the Center list request, *FDBR* then returns the data file to the *listserver* process. At this point, the *listserver* passes the file, untouched, to the *FTM* process. *FTM* completes the data and sends the file off to the *ftm_coproc* process, which in turn returns the file to the *listserver* process. Then, if the request asks only for data more than twelve hours into the past or future, that data will additionally have to be retrieved from the *SDB* process. Alternately, if the request is in the form: REQ OAG…, the *listserver* will make the request *only* of the *SDB* process. When all the data is retrieved from the databases, it is subjected to the filtering queue, filtered flights are counted, and the appropriate report is generated (see Figure 17-4).

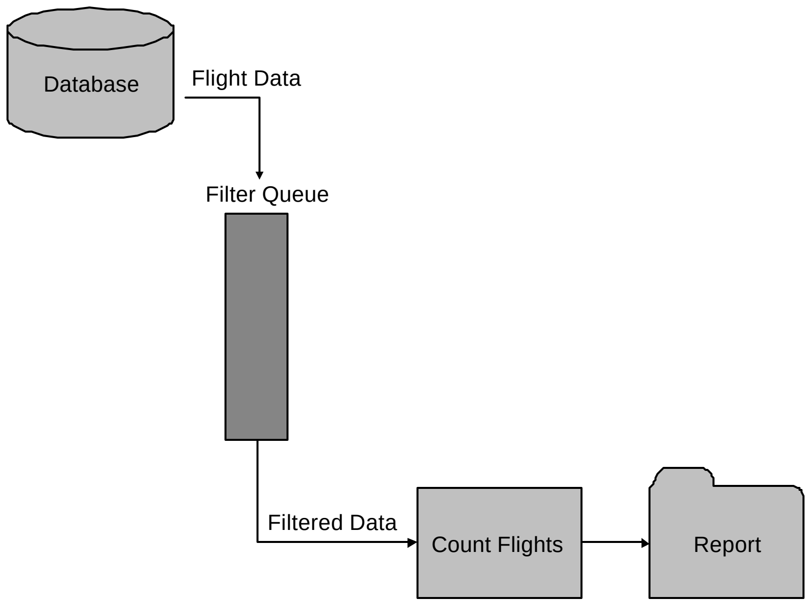

Figure 17-4. Filtering Scheme

Execution Control

The *listserver* is a background process that is started by each node's *nodescan* process that runs on every user_node that an *ASD* runs on. Only one should run on a node, even if there are multiple ASDs/NET.MAILs/TM Shells, etc. Once the *nodescan* process has successfully started a *listserver*, the ASD can hook up to the network address of the *listserver* and does so at initialization. At this point, a user request can be sent via the *ASD* to the *listserver.* It should be noted that programmers can bypass using the *ASD* by making requests directly to the *listserver* through the *NET.MAIL* process or the *TM Shell* process. Statistics requests are also available to the programmer about a *listserver* via the *NET.MAIL process*. **SO** statistics display *Network Address* connections available and **S1** statistics display information about a *listserver's* outstanding requests.

Input

When the *listserver* is started, it creates a list of *Network Addresses* to be used in order to facilitate communication with the various databases. The *listserver* will communicate with ASD, NET.MAIL and *TM Shell* based upon the incoming Network Address of the originator of the request. The *listserver* process treats identically all three types of processes that can originate requests. Once the *listserver* is invoked and initialized, it receives the following input:

- User request **-** this is a command line request for flight information.

- ETMS data - flight information coming from various sources (refer to Figure 17-1).
- Statistics request - this is a command line request for statistics information about the *listserver* module.

Output

The output of the *listserver* process is one of the following reports: flight list, flight counts, sector area reports (AREA), arrival delay prediction (ARRD), arrival fix loading (FIXL), list flight plan (LIFP), scheduled (OAG), or a time verification (VT) report. Output can also take the form of warning or error messages returned to the initiator of the request and of a statistics message returned to *NET.MAIL*. There is also a trace file created in the /etms_path/trace directory that mimics ASD's trace file in naming conventions, various lines initially in the trace file, and in the rules for closing and re-opening a new file with a new date stamp at midnight. Beyond that, the trace file contains what previously appeared if one ran LSTNET in a pad. The name of the trace file is /etms_path/trace/lstnet_transcript.YYYYMMDD.X where YYYY = current year, MM = current month, DD = current day of the month, and .X indicates iteration of invocations that have occurred that day.

Processing Overview

The *listserver* consists of four main modules, depicted in Figure 17-5 as ovals. The four modules are: the *process_lst_req* module, the *parse_req* module, the *process_req* module, and the *ARRD Report Generator* module. The *NODESCAN* process starts up the *listserver* as a background process. Once the *listserver* is initialized, it waits for a request. A request comes in from the *ASD* via *Network Addressing* software, and it is handled by the *process_lst_req* module. The *process_lst_req* module then passes control to the *request_parser* module to decide if the request is syntactically valid.

The *request_parser* interprets the request and in doing so establishes a queue of pointers to routines that will filter the data once it is obtained, flags, and other internal information. After the request is interpreted, the *process_lst_req* sends out a request for data to the sources involved and waits for a response.

When the data is received, it is transferred from the file where it is stored into the data structures used by the *process_req* module or by the *ARRD Report Generator*. At this point either *process_req* or the *ARRD Report Generator* is called, and the data is subjected to the queue of filter pointers (when they are set) and any other conversion/extraction processes. Once the data is processed, it is formatted and written to a file. Finally, the name of this report file is sent back to the *ASD*, through which it can be read (see Figure 17-5).

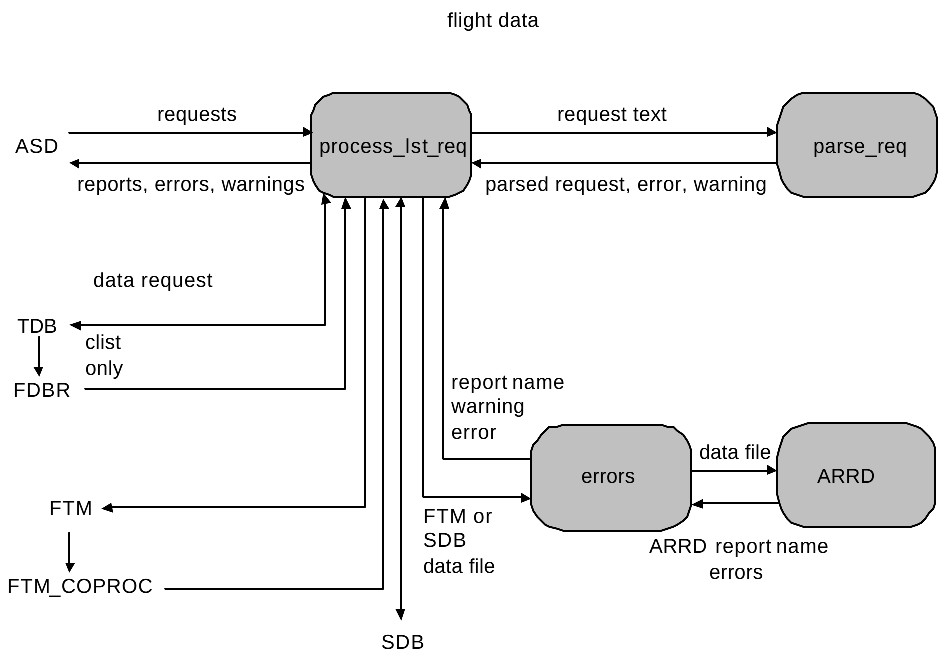

Figure 17-5. Data Flow of the *Listserver* Process

17.1 The process_lst_req Module

Purpose

The *process_lst_req module* is the front-end of the *listserver*. It handles communications with the *ASD* and the various data sources (*FTM*, *SDB*, and *TDB*).

Input

When the *process_lst_req* module is invoked, it receives as input the network addresses of the *ASD* that invoked the request and in the sub-address field, the site number, as defined by the /etms5/shared/config/asd_list_site_data file, and the text of the request itself. The subaddress field is used to determine from which Hub Site one will access the *TDB*, *SDB*, and *FTM* database processes. The request may contain embedded scripts. If so, they are expanded to the full request by the local *listserver* module within the *process_lst_req* module. The *process_lst_req* module needs *ASD*'s network address so that it can properly address mailbox messages to the *ASD* upon completion or the return of any warning or error message. Once it is running, the *process_lst_req* receives data file names from the various data sources when they are queried as well as error and warning messages and status flags from the routines it calls.

Output

The *process_lst_req* module sends the user's request to the *parse_req* module, which once the request is parsed, returns control to the *process_lst_req* module. Then the *process_1st_req* module transfers control to the *process_rec* module, which sends data requests to the various databases. Upon return of control from the *process_req* module, the *process_lst_req* module passes either an error message back to the *ASD* (or the initiator of the request), or a warning and subsequently a report name back to the *ASD* (or the initiator of the request), or the report name back to the *ASD* (or the initiator of the request). For the purpose of simplifying this document, the assumption is made that the initiator of the request is always the *ASD* process.

Processing

As mentioned above, the *process_lst_req* module is the front end of the *listserver* process; i.e., it serves as the interface between the *ASD* and the databases on one side and the *parse_req* and *process_req* on the other. The main task of the *process_lst_req* is to handle communications between these different processes. This task involves keeping track of messages received, messages sent, and communication problems such as queuing of unsent data requests, timing out after waiting for a database response, and unprocessed or semi-processed requests.

Error Conditions and Handling

One of the most common error situations handled by the *process_lst_req* is that of deciding when a communications message is valid and when it is not. The validity of a communications message is determined by a series of tests; once a test fails, the message is discarded.

Besides message validation, the major problem handled by this process is obtaining the data for a request. Errors that arise during this process include not being able to communicate with a particular database or not getting a response once a message is sent. If the unreachable database is the *TDB*, then the request is made to the *FTM*, but only when the location of interest is a pacing airport. Otherwise, the request is removed from the queue, and an error message is sent back to the *ASD*. The request is also aborted when the *SDB* or the *FTM* is unreachable. When a request is aborted, processing begins on the next request in the request queue.

The other major error situation occurs when the *process_lst_req* tries to send a data request, but it cannot because the *Network Addressing software* (i.e., node.sw) is busy. In this case, the data request is queued until the *node.sw* process tells the *process_lst_req* module that it has accepted the request.

17.2 The parse_req Module

Purpose

The *parse_req* is the module that interprets the meaning of the requests and sets up the encoded information in the request to use it for filtering the raw data obtained from the databases. It can also reject a meaningless or syntactically incorrect request. Historically, however, it does little syntax checking and attempts to interpret whatever the user inputs as a valid request.

Input

The *parse_req* has several types of input:

- Demand request as entered by the user (for example **d ord from 1500**)
- Name of the data file, when processing a secondary request

Output

The *parse_req_t* data structure has several types of output:

- Data request information, i.e., the locations and time interval that the user is requesting data for
- Other parsed data, such as airlines or aircraft type to be included or excluded and the type of request (list, counts, AREA, ARRD, FIXL, LIFP, OAG, or VT report)
- Warning status indicator and warning message string, when a warning is to be issued to the ASD process before the report is returned
- Error status indicator and error message string when an error occurs
- Original request **-** the text of the original request (expanded if embedded scripts were originally entered) is stored in a placeholder file under the filename of the request identifier in the /reports directory. When the *process_req* module returns the filtered/formatted file to the *process_lst_req* module, the place-holder file is over-written with the report file. The text of a primary or hub site request are placed as a heading in the primary report output as well. The text of the original request is used as a heading in the report of a secondary request and then a heading follows comprised of the text of the secondary request.

Processing

The main task of the *parse_req* module is to implement the parsing algorithm described in the design issues section; i.e., it takes the user's input string, expands embedded scripts if necessary, and decodes the data in it. It assigns a unique request identifier and a unique request number for this request within this *listserver.* The unique request number is passed to external processes so that if errors are returned, the *listserver* can determine which outstanding request the error is in reference to. For a hub site request the *LIST* and *LSTHUB* processes use their respective request numbers to communicate information about a single request. (Each has its own unique request number to refer to a request.) The request identifier is the file name that the *listserver* creates when done with a request. The *LIST* and *LSTHUB* also have their own unique request identifiers to distinguish a request. Once these assignments have been made and a link is created in the linked list of outstanding requests, the process of understanding the request results in the setting of variables that will enable the data-gathering process and the data processing (filtering and bucketing, etc.). For the most part, these variables fall in the sub-record known as **dsp^.pd.xxx** where **xxx** is the variable being set. DSP (Data Stat Pointer) points you to the correct link in the linked list of requests whose header link is data_stat_head. PD (Parsed Data) refers to information filled in by this *parse_req* module, which assists in requesting the correct information from the Hub processes and FTM and in formatting the report once all the data is returned by the Hub processes and FTMs. Besides interpreting the user's request, the *parse_req* module retrieves data for secondary requests.

Secondary requests are made on the previous data set; these requests usually involve viewing the data in a different way or counting, based on different options. For example, an original request for **a clt 1500 2300** yields a *flight counts* report for all arrivals at **clt** for the time period between **1500** and **2300**. Now, if the user wants to see what the actual flights are and wants them sorted by aircraft ID, the request will look like this: **list sort acid**.

Since the data is already retrieved from the databases, the *parse_req* just obtains the data locally from the data file(s) located in the /rawlist directory. To reduce the response time, the data files were previously returned by the Hub Site processes and FTM. These data files are passed to the *parse_req* module which formats the files, according to the user's new specifications made in the secondary request, and then produces the output report file.

LIST Information Columns, Column Headers, their Keywords, and an Example

The full choice of options a user can specify after the LIST keyword in order to view desired informational "columns" in the output of a LIST report (as opposed to a COUNT report) are:

KEYWORD	Column Header	Contents
ACID/IDENT	ACID	Aircraft Id (flight number) Ex: REQ BOS LIST ACID
ACENTR	ACENTR	Arrival Center Ex: REQ BOS LIST ACID ACENTR
AFIX	AFIX	Arrival Fix Ex: REQ BOS LIST ACID AFIX

AIRL	AIRL	Airline Ex: REQ BOS LIST ACID AIRL
AIRP	ORIG DEST	Origination & Destination Airport Ex: REQ BOS LIST ACID AIRP
AGTD	AGTD	Actual Gate Time of Departure Ex: REQ BOS LIST AGTD

KEYWORD	Column Header	Contents
AGTA	AGTA	Actual Gate Time of Arrival Ex: REQ BOS LIST AGTA
ALT	ALT	Filed Altitude Ex: REQ BOS LIST ACID ALT
RAL	RAL	Last Radar Altitude reported Ex: REQ BOS LIST ACID RAL
ASECT	ASECT	Arrival Sector (last one in sector list) Ex: REQ BOS LIST ACID ASECT
CGTA	CGTA	Controlled Gate Time of Arrival Ex: REQ BOS LIST ACID CGTA
CGTD	CGTD	Controlled Gate Time of Departure Ex: REQ BOS LIST ACID CGTD
CA_DIF	CA_DIF	Diff. between CGTA & AGTA times Ex: REQ BOS LIST ACID CGTA AGTA CA_DIF
CD_DIF	CD_DIF	Diff. between CGTD & AGTD times Ex: REQ BOS LIST ACID CGTD AGTD CD_DIF
CLASS	CLS	Airplane Weight Class (S/L/H) Ex: REQ BOS LIST ACID CLASS
CTG	CTG	Category Ex: REQ BOS LIST ACID CTG
DCENTR	DCENTR	Departure Center Ex: REQ BOS LIST ACID DCENTR
DEST	DEST	Destination Airport Ex: REQ BOS LIST ACID DEST
DSECT	DSECT	Departure Sector (1st in sector list) Ex: REQ BOS LIST ACID DSECT

EFTA EFTA

Ex: REQ BOS LIST ACID EFTA

KEYWORD	Column Header	Contents
ENTRY	ENTRY	Sector Entry Time Ex: REQ BOS LIST ACID ENTRY
EXIT	EXIT	Sector Exit Time Ex: REQ BOS LIST ACID EXIT
ETA	ETA	Estimated Time of Arrival Ex: REQ BOS LIST ACID ETA
ETD	ETD	Estimated Time of Departure Ex: REQ BOS LIST ACID ETD
ETE	ETE	Estimated Time En-route, in minutes Ex: REQ BOS LIST ACID ETE
SPD/SPEED	SPD	Filed Speed Ex: REQ BOS LIST ACID SPD
GS/G_SPD/G_SPEED	GS	Last radar groundspeed Ex: REQ BOS LIST ACID G_SPD
ORIG	ORIG	Origination Airport Ex: REQ BOS LIST ACID ORIG
OGTA	OGTA	Original Gate Time of Arrival Ex: REQ BOS LIST ACID OGTA
OGTD	OGTD	Original Gate Time of Departure Ex: REQ BOS LIST ACID OGTD
OA_DIF	OA_DIF	Diff. between OGTA & AGTA times Ex: REQ BOS LIST ACID OGTA AGTA OA_DIF
OD_DIF	OD_DIF	Diff between OGTD & AGTD times Ex: REQ BOS LIST ACID OGTD AGTD OD_DIF

KEYWORD	Column Header	Contents
PGTA	PGTA	Proposed Gate Time of Arrival Ex: REQ BOS LIST ACID PGTA
PGTD	PGTD	Proposed Gate Time of Departure Ex: REQ BOS LIST ACID PGTD
PA_DIF	PA_DIF	Diff between PGTA & AGTA times Ex: REQ BOS LIST ACID PGTA AGTA PA_DIF
PD_DIF	PD_DIF	Diff between PGTD & AGTD times Ex: REQ BOS LIST ACID PGTD AGTD PD_DIF
RTE	ROUTE	Proposed Route Ex: REQ BOS LIST ACID RTE
AIRWAY	AIRWAY LIST	List of Jet and Victor Routes used Ex: REQ BOS LIST ACID AIRWAY
CENTER	CENTER LIST	List of Centers Traversed Ex: REQ BOS LIST ACID CENTER
FIX	FIX LIST	List of Fixes along route Ex: REQ BOS LIST ACID FIX
SECTOR	SECTOR LIST	List of Sectors Traversed Ex: REQ BOS LIST ACID SECTOR
SGTA	SGTA	Scheduled Gate Time of Arrival Ex: REQ BOS LIST ACID SGTA
SGTD	SGTD	Scheduled Gate Time of Departure Ex: REQ BOS LIST ACID SGTD
SA_DIF	SA_DIF	Diff. between SGTA & AGTA times Ex: REQ BOS LIST ACID SGTA AGTA SA_DIF
SD_DIF	SD_DIF	Diff. between SGTD & AGTD times Ex: REQ BOS LIST ACID SGTD

AGTD SD_DIF

KEYWORD	Column Header	Contents
STAT	no label for field	Status: Proposed, Scheduled, Active, or Controlled Ex: REQ BOS LIST ACID STAT
TGTA	TGTA	ETMS TTM Estimated Gate Time of Arrival Ex: REQ BOS LIST ACID TGTA
TGTD	TGTD	ETMS TTM Estimated Gate Time of Departure Ex: REQ BOS LIST ACID TGTD
TYPE	TYPE	Aircraft Type (i.e.: L101) Ex: REQ BOS LIST ACID TYPE
USER	USR	User Ex: REQ BOS LIST ACID USER
E_TYPE	E_TYPE	Expanded Equip Type (i.e.: 4B/L101/H) Ex: REQ BOS LIST ACID E_TYPE
EQUIP	EQUIP	Equip Prefix/_/Equip Suffix (i.e.: B/_/H) Ex: REQ BOS LIST ACID EQUIP
AC_RMK	Headers as below	All following declared Remarks Flags Ex: REQ BOS LIST ACID AC_RMK
|NRP	NRP	Flag if National Route Program Ex: REQ BOS LIST AC_RMK |NRP
|LIFEGUARD	LFG	Flag if Lifeguard /Medevac Ex: REQ BOS LIST |LIFEGUARD
|CATIII	III	Flag if CATIII Landing Equipment on board (Airports can declare Category III status, means if bad weather can only land if airplane has CATIII equipment Ex: REQ BOS LIST ACID -|CATIII

KEYWORD	Column Header	Contents
|ALTRV	ATV	Flag if 'Altitude Reservation (usu. a military request)' Ex: REQ BOS LIST ACID |ALTRV
|SWAP	SWP	Flag if 'swapping flight from a standard to a non-standard route (usu. due to weather)' Ex: REQ BOS LIST ACID |SWAP
|DIVERT	DVT	Flag if ' diverted to different dest airport' Ex: REQ BOS LIST ACID |DIVERT
|ADCUS	ADC	Flag if should 'Advise Customs' of flight Ex: REQ BOS LIST ACID |ADCUS

The LIST columns that are defaulted if a user only does 'REQ BOS LIST' are as follows for the arrivals section of the BOS report or for a sector report such as REQ ZNY10 LIST:

ACID TYPE ORIG ETD DEST ETA ETE DCENTR

and the defaults for the departure section of the BOS report are:

ACID TYPE ORIG ETD DEST ETA ETE ACENTR

The scheme for spacing between LIST informational columns is as follows. Two spaces will be placed after the right-most character of the previous column or the right-most character of the column header (whichever is wider).

Error Conditions and Handling

This section enumerates the major error situations encountered during the parsing process:

- Unknown word - this is a word that is not contained in the dictionary. The dictionary is also referred to as the codes file. When encountered, a message is sent back to the user with the word in quotes. Processing of that request is then terminated.

- Bad syntax - this error situation occurs when improper options or qualifiers are entered with a keyword. When this error occurs, processing of the current request is terminated, and an error message is sent back to the user.

- No data - this situation is encountered when a secondary request is made, and the input data file is not available to the process.

Warning Conditions and Handling

Warnings can be issued if the request takes a long time or for other reasons. When this occurs, the *request initiator* process is notified with a warning message, but processing of the current request is not terminated.

17.3 The process_req Module

Purpose

The *process_req* is the process that filters the data gathered from the databases based upon variables set in the *parse_req* module. It also counts the data, formats it according to the user's specifications, and then creates the output report file.

Input

The *process_req* module has several types of input:

- Raw data - the flight data obtained from the ETMS databases.
- Parsed request - all the information entered by user request, as well as default information set by the *request_parser* based on the request.

Output

The *process_req* module has the following types of output:

- Report - a flight count, flight list, AREA, ARRD, FIXL, LIFP, OAG, or VT report
- Error status and error message string
- Raw data file - once the report is generated, the *process_req* module stores the data that went into producing the report in a file for use in secondary requests. However, only original request data is stored. The data that goes into a secondary report is not stored, since it is usually a subset of the original request data set, or at best, the same.
- Other parsed information - when handling an original request, all information concerning it is also stored. For example, the time period of the request, the locations for it, whether the data requested was for arrivals, departures, or both, etc. This information is necessary to validate secondary requests.

Processing

The *process_req* module starts its task by running the data through the filters set up by the *parse_req* module in the variables within the record structure, **dsp^.pd**. Once the filtering is done, the flights are counted according to the interval supplied by the user. If nothing else was requested, the *flight counts* report is created. Otherwise, a *flight list* report is formatted and generated. If neither LIST nor COUNT was specified in the request, the default is a *flight counts* report. After the appropriate report is produced, the data that went into it is stored along with any other information pertaining to the request, unless this was a secondary request.

Error Conditions and Handling

The following are the major error situations encountered by the *process_req* module:

- Cannot write to the response data file - the request is aborted, and an error message is sent back to the user.
- Cannot store the raw data - the request is aborted, and an error message is sent back to the user.

17.4 The ARRD Report Generator Module

Purpose

The *ARRD* (ARRival Delay prediction) report returns predicted delays and predicted aircraft stack sizes for a selected time interval at a selected airport. This report is used as a tool by traffic management specialists to monitor near-future traffic flow. The *ARRD Report Generator* uses lists of near-term flight information, airport capacities, and predicted general aviation counts to generate the ARRD reports.

Input

The *ARRD Report Generator* receives the following types of input from the *listserver*:

- Message record - a pointer to a record containing the type of report, the airport, the report start time, the report end time, the stack size, and the stack time
- Flight list - a pointer to a linked list of flights arriving at the specified airport during the specified time
- Capacity list - a pointer to a linked list of capacity values and the times for which they are in effect
- General aviation list - a pointer to a linked list of general aviation counts and the times for which they are in effect
- Output file - the output file name and output file name size, in the form of **pgm_$arg**

Output

The output of the *ARRD Report Generator* is the ARRD report, an example of which can be seen in Figure 17-6. The ARRD report includes nine pieces of data relating to the traffic delays incurred in each hour of the report:

- Arrivals (**ARR**) - the total number of aircraft expected to arrive at the destination airport during each hour of the report plus stack size
- Active flights (**ACT**) - the total number of arrivals for which a departure message is received plus the stack size
- Landings (**LND**) - the predicted number of aircraft landing during each hour of the report
- General Aviation aircraft size (**GA**) - the controller specified number of unscheduled general aviation flights expected each hour
- Capacity size (**CAP**) - the number of flights to be landed during the hour
- Average stack size (**AVERAGE HLD**) - the average number of aircraft projected to be holding in the terminal area for each hour of the report
- Average delay (**AVERAGE DLY**) - the average of projected delays attributed to aircraft holding during each hour of the report
- Peak stack (**PEAK HLD**) - the maximum number of aircraft projected to be holding in the terminal area for each hour of the report
- Peak delay (**PEAK DLY**) - the maximum delay projected for any aircraft holding during the hour

```
        ARRD PIT 03/1700 03/2200

******************************************ARRD  *  REPORT
**************************************
        AIRPORT         : PIT
        START DATE      : May 3
        INTERVAL        :1700 – 2200
********************************************************************************
************************
        PIT ARRIVAL DELAYS 1715

          0 stacked at 0000
                                                  AVERAGE        PEAK
        TIME    AR     AC     LN     CA     G      HLD     DL     HL     DL
                 R      T      D      P     A               Y      D      Y
        170     32      8     32     56     8       0       1      4      4
         0
        189     22      8     21     56     8       0       1      4      4
```

Figure 17-6. Sample ARRD Report

Processing

In general, the *ARRD Report Generator* creates *ARRD* reports in the following manner. The *ARRD Report Generator* first sorts the flight list received from the *listserver* and translates the times from UTC to GMT. Using the capacity of the selected airport (expressed as the number of aircraft landings per hour), the *ARRD Report Generator* divides each hour of the simulation into landing slots. The time difference between these slots is equal to the sixty minutes in the hour divided by the capacity. The *ARRD Report Generator* steps through the arrival demand list (flight list) assigning flights to landing slots and gathering arrival delay data, based on these assignments. After sequencing through all of the flights in the list, the data collected is summed and averaged to produce the hourly statistics.

Each flight in the flight list is assigned the earliest available landing time that is equal to or greater than the flight's Estimated Time of Arrival (ETA) until the total number of landing slots for the time period is exhausted or until the end of the flight list is reached. Landing slots become unavailable when assigned to an arriving flight, and any landing slots before the last assigned slot become unavailable whether used or not.

In order to collect the counts needed to determine the peak and average stack size, the *ARRD Report Generator* has a set of sixty counters, corresponding to the minutes of the hour. If a flight must hold (i.e., the assigned landing time is later than the estimated time of arrival), every minute counter, which corresponds to the time span bounded by and including the ETA and the assigned landing time, is incremented. After all the flights in the list are exhausted, the counters corresponding to the respective hours are summed and averaged. The maximum of these counts is the peak stack size for the hour.

In gathering the hourly delay data, all delays associated with each flight are added to the cumulative totals for the respective hours. For those aircraft that began their hold in previous hours, the entire delay minus any holding time prior to this hour is used. For example, an aircraft that is projected to have a three-hour delay and arrives at 1230 would contribute a three-hour delay to the 1200 hour statistics, a two-and-a-half-hour delay to the 1300 hour, a one-and-a-half hour delay to the 1400 hour, and a half-hour delay to the 1500 hour. The average delay per hour is the sum of the cumulative delays divided by the number of flights contributing to those delays. The largest projected delay occurring in each hour is the peak delay.

In certain cases, the user may set certain conditions under which the *ARRD Report Generator* will simulate airport arrivals. Besides setting a capacity value, the user might also set the number of general aviation flights. If not set, the default of eight is used. Dummy general aviation flights are distributed throughout the hour when the number of active flights in the hour is less than the general aviation count specified. In addition, the user may denote a stack time and a stack size. If specified, dummy flights are added to the flight list at the designated stack time and are assigned landing slots accordingly.

Error Conditions and Handling

When an error occurs, the *ARRD Report Generator* returns to the calling program (*listserver*).

17.5 Listserver Source Code Organization

This section describes the source code used in building the executable version of the *listserver*. The source code resides in Pascal files. Each file contains one or more functional units called a routine. A routine is implemented as either a Pascal function or procedure. The Pascal files have been organized as elements in two Domain System Engineering Environment (DSEE) libraries. The *listserver_lib* contains the source code for the *process_lst_req*, *parse_req*, and the *process_req* modules. The source code for the *ARRD* module is contained in the *arrd_lib* library.

In order to build the *listserver*, the following files should exist in your work directory:

- **~/work/lstsrv.thread.model**
  The contents of the lstsrv.thread.model file should read as follows:
  - -reserved
  - [lstnet.v.05.75] -when_exists
- **~/work/lstsrv.thread**
  To build a non-debug version, the contents of the lstsrv.thread file should read as follows:
  - -FOR ?*.pas -USE_OPTIONS -opt 0
  - -reserved
  - [lstnet.v.05.75] -when_exists

  To build a debug version, the contents of the lstsrv.thread file should read as follows:
  - -FOR ?*.pas -USE_OPTIONS -opt 0-config dbg -config timing -dba
  - -reserved
  - [lstnet.v.05.75] -when_exists

In order to build the *listserver*, the following DSEE commands must be issued:

- **set system listserver_sys**
- **set lib listserver_lib**
- **set thread -model lstsrv.thread.model**
- **set model lstsrv.sml**
- **set thread lstsrv.thread**

- **build**
- **name version test_listserver! lstsrv.v.05.75**
- **exp listserver! -into /etms5/list/lstnet -r**
- **OR... instead of (7) and (8) above, execute single DSEE command below: cre rel lstnet.v.05.75 -fro test_listserver! -exp listserver! -into /etms5/list/lstnet -r**

17.6 Listserver Data Structure Tables

Tables 17-1 through 17-3 describe the data structures.

17.6.1 The data_stat_t Data Structure

The **data_stat_t** data structure holds all variables associated with a single request. It defines one link in the linked list of outstanding requests in the *listserver* process. The pointer to the front of this linked list is called **data_stat_head**. This pointer points to the "front" of the actual linked list whose name throughout the *listserver* process is **dsp^**. All parsed request data, time interval data, and flight data reside within each of these links. This record contains all the data fields that the user has access to throughout the *listserver* (see Table 17-1).

17.6.2 The parsed_data_t Data Structure

The **parsed_data_t** data structure contains returned variables filled in in order to make the data requests from the database processes and in order to filter requests later when the database processes return all the data. The **parsed_data_t** variables are filled in by the *parse_req* module as depicted above. The variables contained in this structure are outlined below (see Table 17-2).

17.6.3 The flight_rec_t Data Structure

The **flight_rec_t** data structure contains returned data from the database processes about the requested flights in the *listserver* process. All flight data obtained from any of the databases is loaded into linked lists of records of **flight_rec_t** type. This record contains all the data fields that the user has access to throughout the *listserver* (see Table 17-3).

Table 17-1. data_stat_t Data Structure

data_stat_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
req_id	Unique Request Identifier; (i.e., Report Name)	loc of interest, lst or cnt, date, time	-	string22
req_num	Unique Request Number	used by interpro-cess commun.	1 – 999	integer16
pd	Parsed Data; record containing all returned by parse_req mod.	-	-	parsed_data_t
sender	Network Address of initiator of request	Network Address	-	net$_decoded_ header_t
got_tbd_dat	Flag indicating if TDB data rec'd for the request yet	-	yes/no	boolean
tbd_start_t	Start date/time of data requested form TDB	-	-	cal_$timedate_rec_ t
tbd_end_t	End date/time of data requested from TDB	-	-	cal_$timedate_rec_ t
sdb_start_t	Start date/time of data requested from SDB	-	-	cal_$timedate_rec_ t
sdb_end_t	End date/time of data requested from SDB	-	-	cal_$timedate_rec_ t
go_to_ftm	Flag indicating if data needed from FTM for this request	-	yes/no	boolean
need_sdb_dat	Flag indicating if data requested from SDB for this request	-	yes/no	boolean
need_tdb_dat	Flag indicating if data needed from TDB for this request	-	yes/no	boolean
need_ftmfill	Flag indicating if holes in data requested by FTM; need to fill	-	yes/no	boolean
need_ftp_dat	Flag indicating get ACK from FTP on hub site requests only	-	yes/no	boolean
need_ftm_dat	Flag indicating if data needed from FTM for this request	-	yes/no	boolean
need_list_dat	Flag indicating if data needed from LSTHUB for this request	-	yes/no	boolean
reqst_sdb_only	Flag indicating user wants only SDB data (i.e., REQ OAG…)	-	yes/no	boolean
xmit	in LIST w/hub req, Flag is true if request sent up to LSTHUB	-	yes/no	boolean
hub_reply	Flag indicating LSTHUB process is waiting to reply to LIST	-	yes/no	boolean
wait_for_xmit	True if LSTHUB waiting on ACK from FTP that LIST got report	-	yes/no	boolean

next	Pointer to next request in linked list of outstanding requests	-	-	data_stat_prt_t

Table 17-2. parsed_data_t Data Structure

parsed_data_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
acid_lst	List of Aircraft Identifiers in the request	-	-	concat_lst
airp_lst	List of Airport Identifiers in the request	-	-	concat_lst
arrd_info	Data requested/returned by the ARRD module	-	-	arrd_ptr_t
category	Contains user specified COUNT options to filter report by	record	-	category_rec_t
data_source	Array of all the possible data sources that a request may need	array	1…db_type	array of com_array_type
last_data_source	Last data source sent a request for info to this list request	-	1…db_type	db_type
demand	Req. type desired: A=Arrivals, D=Departures, or B=Both	-	A, D, or B	req_demand_t
db_rep_type	Kind of report desired: AIRP, Sector, Center, FIX, LIFP,etc.	-	-	db_req_rec_t
field_cnt	Count of fields request.	-	-	integer16
filter_count	Count of filters to apply when data is returned by sources	-	-	integer16
filter_q	Array of addr of filter routines to be applied after data returned	-	1…max_filter	array of func_ptr_t
fixlist	List of fixes assoc with airport in loc_interest list	-	-	fixptr
fixl_recvd_time	Time fix loading message received	-	-	integer16
flname	Hubsite or secondary request's filename	-	-	pgm_$arg
init_flname	Hubsite requests storage for initiator's request full filename	-	-	prm_$arg
req_interval	Default minutes to bucket output report into.	-	Default 15	integer16
req_txt	Character array representing request text input by user.	-	-	string1792
req_len	Length of input request contained in req_txt above.	-	-	integer16
resp_flname	Unique name and length of the report pathname to generate.	-	-	pgm_$arg
sort_field	integer indicating if sort on ARR, DEP, ARR – TIME, DEP – TIME	-	-	integer16

state	State representing the part of request presently parsing	req list cnt fix lfp	ard plt fct lsto	state_t
target_fix	If loc_interest_inc_l contains ARR_FIX set & find ARR_AIRP.	-	-	string6

Table 17-2. parsed_data_t Data Structure (continued)

parsed_data_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
title_count	Unsued variable.	-	-	integer16
tv	Time Verification Information, if requested.	-	CT,OT,PT,ST	integer16
num_remote_tries	Number of time process has tried to contact another process.	-	-	integer16
count_num_centers	Number of centers in the location of interest inclusion list.		-	integer16
ucap	User entered capacity value.	-	-	cap_ga_ptr
cap	TDB dictated capacity value.	-	-	cap_ga_ptr
ugas	User entered General Aviation Estimate value.	-	-	cap_ga_ptr
gas	TDB dictated General Aviation Estimate value.	-	-	cap_ga_ptr
a_stats	Arrival buckets. A bucket contains all for an arr. time int.	-	-	stat_bucket_ptr
d_stats	Departure buckets. A bucket contains all for a dep time int.	-	-	stat_bucket_ptr
calc_ftv	Calculate Flight Time Verification (Early, On-Time, Late)	VT key word	T/F	boolean
clist	Is this request a center list request? (Yes means True.)	-	T/F	boolean
do_fixl	Is this request a Fox Loading request? (Yes means True.)	-	T/F	boolean
need_header	Flag stating whether the report should have a header (request).	-	T/F	boolean
print_area	This flag is True if this is an AREA request; False otherwise.	-	T/F	boolean
print_cnts	This flag is True if COUNTS report; False if LIST report.	Default True	T/F	boolean
print_fix	True if this request has the fix key word in the request.	-	T/F	boolean
print_flts	Print entire Arrival Departure or both sections of report.	-	T/F	boolean
print_stats	Print Arr/Dep Time verification Statistics (Early, On – Time, Late)	-	T/F	boolean
xmit	True if this is a hubsite request.	-	T/F	boolean
remote_req_id	If this is LSTHUB process/req., contains the LIST report name.	-	-	string22

Table 17-2. parsed_data_t Data Structure (continued)

parsed_data_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
remote_req_num	Unique request# within remote LIST process distinguishing req.	-	-	integer16
initiator	Network Address of the initiator of the LIST request.	-	-	net_$address_t
hub_proc	Network Address of the LSTHUB that recvd. hub request	-	-	net_$address_t
hub_reply	In remote LIST means waiting ona hub LSTHUB response.	-	T/F	boolean
wait_for_xmit	Wait far a transmit operaion to complete.	-	T/F	boolean
full_asd_resp_flname	Fully qualified pathname for output file at remote LIST site.	-	-	pgm_$arg
using_old_data	If secondary request then one is using old data from/rawlist dir	-	T/F	boolean
secondary_req	True if the user issued a secondary request.	-	T/F	boolean
format_options_q	List of user specified options the report will be formatted via.	array	1…maxopt	format_opt_t
count_time	If COUNT report, Count report time range.	-	-	count_time_rec_t
filter_time	Tracks time type used in filtering, i.e., SGTA, PGTA, etc.	-	-	filter_time_t
list_time	If AREA report, then time range for the report desired.	-	-	timedate_rec_t
req_time	Time range for which the request was issued.	-	-	timedate_rec_t
start_utc	Request start date/time in whole seconds in Univ. Coord. Time	-	-	integer32
end_utc	Request end date/time in whole seconds in Univ. Coord. Time	-	-	integer32
today	Today as perceived by parsing algorithm to expand date/time.	-	-	cal_$timedate_t
count_filter_q	Array of count filters to apply	array	1…4	func_ptr_t
count_stack_size	Number of count filters on the count_filter_q (above)	-	-	integer16
maxlinelen	Report output line size. 85 if not overridden via SETLEN keyword	-	-	integer16
ac_cat_inc_l	List of user specified aircraft categories to include.	-	-	wdptr_t

ac_cat_exc_l	List of user specified aircraft categories to exclude.	-	-	wdptr_t

Table 17-2. parsed_data_t Data Structure (continued)

parsed_data_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
class_inc_l	List of user specified Aiecraft Classes to include.	-	S, L, H	wdptr_t
class_exc_l	List of user specified Aircraft Classes to exclude.	-	Small, Large, Heavy	wdptr_t
user_inc_l	List of COUNT USER Aircrafts user specifies to include.	-	T, F, C, G, M	wdptr_t
user_exc_l	List of COUNT USER Aircrafts user specifies to exclude.	-	T, F, C, G, M	wdptr_t
aircraft_inc_l	List of user specified aircraft types to include.	-	i.e., L101	wdptr_t
aircraft_exc_l	List of user specified aircraft types to exclude.	-	i.e., -B727	wdptr_t
airlines_inc_l	List of user specified airline(s) to include.	-	i.e., TWA	wdptr_t
airlines_exc_l	List of user specified airline(s) to exclude.	-	i.e., -UAL	wdptr_t
count_loc_inc_l	If COUNT report, user specified Location(s) of Interest to include.	-	-	wdptr_t
count_loc_exc_l	If COUNT report, user specified Location(s) of Interest to exclude.	-	-	wdptr_t
loc_interest_inc_l	List of user specified Primary Location of Interest to include.	-	-	wdptr_t
loc_interest_exc_l	List of user specified Primary Location of Interest to exclude.	-	-	wdptr_t
count_option_list	List of Count/Category columns for Inclusion/Exclusion in report.	-	-	wdptr_t
sort_option_list	List of Columns by which to sort report.	-	Chronological Def.	wdptr_t
info_option_list	User specified columns desired if LIST report (vs. COUNT).	-	-	wdptr_t
sort_key	User specified SORT criterion.	-	-	wdptr_t
restrict_loc_exc_l	Secondary Location(s) of Inclusion List	-	-	wdptr_t
restrict_loc_inc_l	Secondary Location(s) of Inclusion List	-	-	wdptr_t
afix_inc_l	List of user specified Arrival Fix(es) to include.	-	-	wdptr_t

afix_exc_l	List of user specified Arrival Fix(es) to exclude.	-	-	wdptr_t
sector_inc_l	List of user specified Sector(s) to include.	-	-	wdptr_t

Table 17-2. parsed_data_t Data Structure (continued)

parsed_data_t				
Library Name: listserver_lib		**Purpose:** This record contains all information for a single request.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
sector_exc_l	List of user specified Sector(s) to exclude.	-	-	wdptr_t
fix_inc_l	List of user specified Fix(es) to include.	-	-	wdptr_t
fix_exc_l	List of user specified Fix(es) to exclude.	-	-	wdptr_t
center_inc_l	List of user specified Center(s) to include.	-	-	wdptr_t
center_exc_l	List of user specified Center(s) to exclude.	-	-	wdptr_t
airway_inc_l	List of user specified airway(s) to include.	-	-	wdptr_t
airway_exc_l	List of user specified airway(s) to exclude.	-	-	wdptr_t
equip_inc_l	List of user spec. Aircraft Equip. Prefix and Suffixes to include.	-	-	wdptr_t
equip_exc_l	List of user spec. Aircraft Equip. Prefix and Suffixes to exclude.	-	-	wdptr_t
ac_rmk_inc_l	List of user spec. Aircraft Remarks to include.	-	-	wdptr_t
ac_rmk_exc_l	List of user spec. Aircraft Remarks to exclude.	-	-	wdptr_t
ac_rmk_fmt_inc_l	List of user spec. AC_RMK columns to include in a list report.	-	-	wdptr_t
ac_rmk-fmt-exc_l	List of user spec. AC_RMK columns to exclude from a list report.	-	-	wdptr_t
e-type_inc_l	List #acft/EQP_PREFIX/TYPE/ EQP_SUFFIX to include.	-	-	dbl_wdptr_t
e_type_exc_l	List #acft/EQP_PREFIX/TYPE/ EQP_SUFFIX to exclude.	-	-	dbl_wdptr_t
fixl_bucket_list	List of buckets for a FIXL report. (Fix Loading Report)	-	-	bucket_ptr
arr_bucket_list	List of buckets to separate the arrival portion of a report into.	-	-	bucket_ptr
dep_bucket_list	List of buckets to separate the departure portion of a report into.	-	-	bucket_ptr

Table 17-3. flight_rec_t Data Structure

flight_rec_t				
Library Name: listserver_lib		**Purpose:** This record contains all the available information for a flight.		
Element Name: parser.ins.pas				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
flight_id	Aircraft identification	1 or more letters followed by digits	-	string7
dep_ap	Departure Airport	FAA Airport designator	-	string5
sched_dep_time	Scheduled departure time	UTC	-	integer32
prop_dep_time	Proposed departure time	UTC	-	integer32
ttm_dep_time	TTM modeled time	UTC	-	integer32
cntrl_dep_time	Controlled departure time	UTC	-	integer32
actual_dep_time	Actual departure time	UTC	-	integer32
est_dep_time	Estimated departure time	UTC	-†	integer32
orig_dep_time	Original departure time	UTC	-	integer32
dep_t_type	Best departure time available	1 char designator	A, P	char
ete	Estimated time enroute	minutes	1 – 1440	integer16
element_name	Sector/Fix that put flight into response from TDB/FTM.	-	-	array[1…10] of char
entry_time	Sector entry time	minutes since midnight	0 – 1439	integer16
exit_time	Sector exit time	minutes since midnight	0 – 1439	integer16
ttm_slot	TTM flight time slot	Hour since midnight	0 – 23	integer32
arr_ap	Arrival airport	FAA airport designator	-	string5
sched_arr_time	Scheduled arrival time	UTC	-	integer32
prop_arr_time	Proposed arrival time	UTC	-	integer
ttm_arr_time	TTM modeled time	UTC	-	integer32
cntrl_arr_time	Controlled arrival time	UTC	-	integer32
actual_arr_time	Actual arrival time	UTC	-	integer32

est_arr_time	Estimated arrival time	UTC	-†	integer32

Table 17-3. flight_rec_t Data Structure (continued)

flight_rec_t				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
orig_arr_time	Original Arrival Time	UTC	-	integer32
arr_t_type	What is the best estimate for the arrival time	single char	A, E	char
arr_fix	Arrival fix	FAA Fix designator	-	string6
arr_fix_time	Time at the arrival fix	UTC	-	integer32
air_car	Airline name	FAA airline designator	-	string3
ac_cat	Aircraft category	single char	J, P, T	char
ac_class	Aircraft class	single char	H, S, L	char
num_aircraft	Number of Aircraft (flying in formation on a single flight plan)	single digit	1 – 9	char
ac_eqp_prefix	Aircraft Equipment Prefix	[A – Z]/	A – Z	char
ac_type	Aircraft type	FAA aircraft type designator	-	string4
ac_eqp_suffix	Aircraft Equipment Suffix	/[A – Z]	A – Z	char
active	Is this an active flight?	-	yes/no	boolean
alt	Flight's Radar altitude (last reported)	100's of feet	-	integer16
ascii_altitude	ASCII for Radar Altitude (i.e., has Block altitude within)	in 100's of feet xxxByyy or xxx	-	string8
f_alt	Flight's Filed Altitude	100's of feet	-	integer16
ascii_f_alt	ASCII for Filed Altitude (i.e., has Block altitude within)	in 100's of feet xxxByyy or xxx	-	string8
alt_type	Altitude type of the radar alt only	[A – Z]	-	char

Table 17-3. flight_rec_t Data Structure (continued)

flight_rec_t				
Data Item	**Definition**	**Unit/Format**	**Range**	**Var. Type/Bits**
speed	Radar Ground Speed	-	-	integer16
f_speed	Filed Speed	-	-	integer16
user_cat	User category	single char	C, F, G, M, T	char
militar	Is this a military flight?	-	yes/no	boolean
diffs	Used to compute time differences for VT reports	-	-	vt_dif_typ
route_len	Length of the route string	-	-	integer
route	Route	char string		string256
remarks_flags	16 bits of Boolean values as to whether remark is on/off	bitflags	0 – 1 per bit	integer16
num_sectors	No. of sector names in sector list Max. of 50 as in FTM	-	0 – 50	integer16
sector_list	List of 6-byte sector names	6 bytes each	max 50 in list	string300
num_fixes	No. of fixes in fix_list Max. of 50 as in FTM	-	0 – 50	integer16
fix_list	List of 6-byte fix names	6 bytes each	max 50 in list	string300
num_airways	No. of airway names in airway list Max. of 50 as it FTM	-	0 – 50	integer16
airway_list	List of 6-byte airway names	6 bytes each	max 50 in list	integer300
num_centers	No. of centers in center_list Max. of 10 as in FTM	-	0 – 10	integer16
center_list	List of 3-byte center names	3 bytes each	max 10 in list	string10

† The estimated departure time is labeled with an A when the flight is flagged as being active. Otherwise, it is labeled with a P. The estimated arrival time is labeled with an A when the time corresponds to the actual arrival time. Otherwise, it is labeled with a P.